04-三步走：如何了解一个软件的设计？

你好！我是郑晔。

经过了前面几讲的铺垫，我们已经对软件设计是什么，以及要考虑哪些因素有了一个初步的了解。热身之后，就该开启正式的旅程了。

作为一个程序员，我们在职业生涯中免不了要接手新项目，承担维护该项目的职责。如果一个新项目摆在面前，你会怎么去研究它呢？

很多人的第一反应就是去看源代码。但是，一头扎入代码中，很快你就会迷失其中，最初那股子探索精神，也会逐渐被迷茫所代替。回想一下，有多少次你满怀希望地打开一个开源项目，结果多半都是坚持不了多久就放弃了。你有没有想过，问题出在哪里呢？

你的迷茫在于缺少对这个软件整体的了解，这就如同不带地图指南针闯入密林一般，迷路只是早晚的事。所以，虽然阅读源码是必经的一步，却不应该是你的第一步。我们应该先从了解软件的设计开始。那我们该如何了解一个软件的设计呢？

模型、接口和实现

了解一个软件的设计可以从三个部分着手：**模型、接口和实现**。这三者的关系就好比你去看代码，你会先去看有哪些类以及它们之间的关系，这就是看模型；然后你会打开一个具体的类，看它提供了哪些方法，这就相当于看接口；最后，你再来打开一个具体的方法，去看它的代码是怎么写的，这就是看实现。

好，接下来，我们具体地分析一下每一个部分。

首先是**模型**，它是一个软件的核心部分。在其它的材料里，也有人称之为抽象，为了统一，我这里就都叫模型了。我们在前面的课程里也说过，设计最关键的就是构建出模型。而理解一个设计中的模型，可以帮助我们建立起对这个软件整体的认知。

比如，你在编写分布式计算代码时，需要考虑怎样在不同的节点上调度计算；而使用MapReduce时，只要考虑如何把计算分开（Map）最后再汇总（Reduce）；而到了Spark，注意力就集中在要做怎样的计算上。它们在解决同样的问题，只是抽象层次逐步提高了，越来越接近要解决的问题，越来越少地考虑计算，在不同的机器上是如何执行的，由此，降低了理解的门槛。

当你知道了模型的重要性，目光甚至可以不局限在某一个软件上。如果把同一个领域不同阶段的多个模型联系起来看，你还能看到软件发展的趋势。

其次是**接口**，它决定了软件通过怎样的方式，将模型提供的能力暴露出去。它是我们与这个软件交互的入口。如何理解这句话呢？我给你举几个具体的例子。

- 一个程序库的接口就是它的API，但对于同样的模型，每个人会设计出不同的API，而不同的API有着不同的表达能力。比如：Google的Guava对JDK的一些API重新做了封装，其目的就是为了简化开发，而很多优秀的做法后来又被JDK学了回去。
- 一个工具软件一般会提供命令行接口，比如，每个程序员必备的基本技能——Unix命令行工具就是典型的命令行接口。

- 一个业务系统的接口，就是对外暴露的各种接口，比如，它提供的各种REST API，也可能是提供了RPC给其它系统的调用。
- ……

如果你想深入源码，去了解一个软件，接口是一个很好的指向标。你可以从一个接口进入到软件中，看它是怎样完成各种基本功能的。

最后是**实现**，就是指软件提供的模型和接口在内部是如何实现的，这是软件能力得以发挥的根基。这么说可能比较抽象，我再来举些例子。

- 一个业务系统收到一个请求之后，是把信息写到数据库，还是转发给另外的系统。
- 一个算法的实现，是选择调用与别人已有的程序库，还是需要自己实现一个特定的算法。
- 一个系统中的功能，哪些应该做成分布式的，哪些应该由一个中央节点统一处理。
- 一段业务处理，是应该做成单线程，还是多线程的。
- 当资源有竞争，是每个节点自己处理，还是交由一个中间件统一处理。
- 不同系统之间的连接，该采用哪种协议，是自己实现，还是找一个中间件。
- ……

讲到这，相信你一定发现了，“实现”里面的内容很多。所以，做每一个技术决策都应该结合自己所开发应用的特点，并不存在一个通用的解决方案。在实际的工作中，我发现许多人以为的设计其实是这里所讲的实现。

我也知道，“实现”很重要，但是，它必须建立在模型和接口的基础之上。因为在一个系统的设计中，模型是最核心的部分。如果模型变了，这个软件便不再是这个软件了，而接口通常反映的就是模型。所以，模型和接口的稳定度都要比实现高，实现则是要随着软件发展而不断调整。

举个例子，很多人都知道Redis这个键值对存储性能非常好，他们学习Redis时，对其单线程模型印象深刻，因为它简单高效。但随着人们使用Redis的增多，对Redis有了进一步的需求。所以，从6.0开始，它开始支持多线程版本，以便于更好地满足人们的需求。但即便Redis改成了多线程，它还是那个Redis，它的模型和接口还是一如既往，只是实现变了而已。

了解设计三步走

之所以要把模型、接口和实现区分开来，是因为这三者的关注点是不同的，而很多人在讨论所谓的“设计”时，经常会把它们混在一起。

如果你在讨论的时候连“讨论的内容到底是什么”都没弄清楚，就很难得出一个清晰的结果。我参与过很多类似的讨论，经常有一种很混乱的感觉。我思考了很长时间才发现，问题就在于他们把不同层面的内容混在了一起。

所以正确的做法是什么呢？就是你在讨论设计时应该遵循一个顺序，**先模型，再接口，最后是实现**，同理，了解一个设计也应该遵循这样的顺序。

如果模型都还没有弄清楚，就贸然进入细节的讨论，你很难分清哪些东西是核心，是必须保留的，哪些东西是可以替换的。如果你清楚了解了模型，也就知道哪些内容在系统中是广泛适用的，哪些内容必须要隔离。简单地说，分清模型会帮助你限制实现的使用范围。

下面是一张简化过的架构图，在这幅图里，订单模块完成处理之后，通过一个Kafka队列把消息发给支付模块，支付模块处理之后，再通过一个Kafka队列把消息发给物流模块。很多人都应该在自己的项目中见过类似的，但是更复杂的架构图。你能看出这张图的问题在哪吗？

这张架构图的问题就在于，它把模型和实现混淆在一起了。图中的订单、支付和物流，说的都是模型层面的东西，但Kafka的出现，就把实现层面的东西拉了进来。Kafka只是实现这个功能时的一个技术选型，这也就意味着，如果随着业务的发展，它不能很好地扮演它的角色，你就可以替换掉它，而整个设计是不用变的。

所以，实现这段代码的时候，必须把Kafka相关的代码进行封装，不能在系统各处随意地调用，因为它属于实现，是可能被替换的。

我还要强调一点，在了解设计时，要按层次去了解，因为设计常常是分层的。每当我们打开一个层次，需要了解它的内部时，我们还要按照模型、接口和实现的顺序解读这个层次。

我用大家比较熟悉的操作系统来举个例子，如果你去了解它的内部，就知道它有内存管理、进程调度、文件系统等模块。我们可以按照模型、接口和实现去理解每个模块，就以进程管理为例：

- 进程管理的核心模型就包括进程模型和调度算法；
- 它的接口就包括，进程的创建、销毁以及调度算法的触发等；
- 不同调度算法就是一个个具体的实现。

操作系统课程难以学习，很大程度上就在于，很多人没有搞清楚其中各个概念之间的关系。

即便层层展开到最后，到了一个具体类，甚至是一个具体的数据结构，我们依然可以按照模型、接口和实现这个结构来理解，比如很多Java面试题常问到的HashMap：

- 它的模型就是我们在数据结构中学习的HashMap；
- 它定义了一些接口，比如，get、put等；
- 它的实现原来是用标准的HashMap实现，后来则借鉴了红黑树。

实际上，当你能够一层一层地去理解设计，就像一棵知识树逐渐展开一样，每一个知识节点在展开的时候，都会有下面一级更具体的内容。当你的头脑中有了这样一棵设计树，你也就掌握了整个系统的地图，再有新需求到来时，你就不会再盲目地去改代码了。

总结时刻

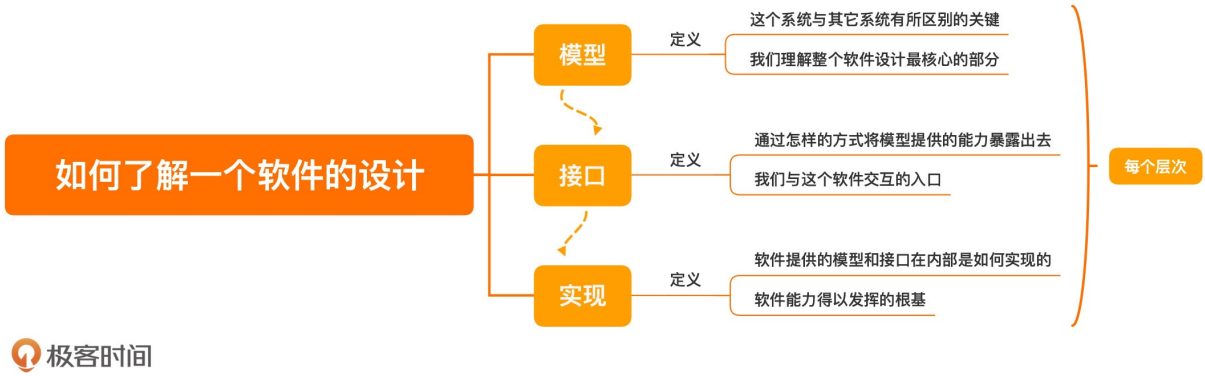
今天，我们学习了如何了解一个软件设计，可以从三个部分入手：模型、接口和实现。

- 模型，也可以称为抽象，是一个软件的核心部分，是这个系统与其它系统有所区别的关键，是我们理解整个软件设计最核心的部分。
- 接口，是通过怎样的方式将模型提供的能力暴露出去，是我们与这个软件交互的入口。
- 实现，就是软件提供的模型和接口在内部是如何实现的，是软件能力得以发挥的根基。

了解设计的顺序应该是，**先模型，再接口，最后是实现**。了解设计，需要一层一层地展开，在每个层次都按照模型、接口和实现进行理解，在头脑中形成一棵设计树。

现在你已经有了一个了解设计的基本方法，接下来几讲，我会用几个开源项目带你再进一步，去看看如何去了解模型、接口和实现。

如果今天的内容你只能记住一件事，那请记住：**了解设计，先模型，再接口，最后是实现**。



思考题

现在的开源项目越来越多，每个开源项目都会提供一些不同的特点，请你找一些自己感兴趣的开源项目，看看它们分别提供了什么，是新的模型、是新的接口，还是新的实现？欢迎在留言区分享你的思考。

感谢阅读，如果你觉得这一讲的内容对你有帮助的话，也欢迎把它分享给你的朋友。

精选留言：

- 一步 2020-06-01 18:39:46
  当使用一个新的 库或者框架的时候 首先看的是 接口，看对外提供的功能的是否满足自己的要求，然后才是 具体的实现。 对于模型 想学习开源软件的架构的时候在去关注的 [2赞]

- 业余爱好者 2020-06-01 12:14:36
  使用一个软件，就是通过其接口进行的。接口分为api和ui。要想正确使用一个东西，就要知道一点内部的原理。模型是对内部原理的简化，可以让人快速上手。如果想玩一些高级的玩法，或者想要改造软件，就少不了研究实现了。

  接口和模型组合起来，就相当于一个“ADT”。 [1赞]

- FelixFly 2020-06-01 10:01:34
  这个分析思路比较好，好多一上来就深入源码细节（实现）容易懵，特别是集成关系比较重的时候，最后都不知道调用到哪地方去了，这样就需要单步跟踪才能了解其脉络。若是先分析了类的层次关系，有个一定的理解，后面的调用脉络就有一定的清晰认识。模型这部分比较难理解，要想到作者为什么会这样定义模型，这样的设计用途是什么，可能刚开始不了解，到实现部分有种豁然开朗的感觉。 [1赞]

  作者回复2020-06-02 08:26:49
  有了地图，才不迷路。

- escray 2020-06-01 09:51:06
  看了文章，知道自己以前为什么读不懂源码了，因为我每次都是从“实现”入手，想要自顶向上的去了解整个软件的架构，也许这条路也能走通，但是难度太大。

  之前看过《代码阅读方法与实践》（和大多数书一样没能看完），这本书其实就是从细节开始讲起的，书是好书，但是我的打开方式似乎有点问题。

  进程管理的模型包括调度算法，这个稍微有一点不好理解，我之前以为模型都是一些比较“实”的东西。不过如果从模型是“名词” / 概念，接口是“动词” / 命令，实现是动作 / 技术细节这个角度就更清楚一些。

翻回到专栏的第一篇，老师举例说在交易系统中，有交易原语：下单、成交和撤单，交易动作：冻结、解冻、出金、入金，当时解释说这是模型上的分层。那么我的问题是，这些原语和动作都当做模型来看待么？

如果让我来分析，那么交易系统的模型可能是：用户、商品、订单、支付、物流；接口是：下单、成交、撤单……

看到 @Kǎfκã²º²º 的留言，他是把“回款”也放到模型里面了。

似乎是因为老师抽象的层次更高一点，期待下一篇关于模型的讲解。 [1赞]

- 阳仔 2020-06-01 07:37:46
理解软件开发中的设计可以通过三步走套路:
模型-接口-实现，
这个也就是了解软件设计的“模型”。
模型其实是一个软件设计中的抽象，通过与其它软件设计中的进行对比学习，理解它们的异同，突出自身的核心抽象结构;
接口是软件交互的入口，是一个软件系统的能力提现，它是一种规范，它与模型共同组成了软件系统的稳定性因素;
实现是软件设计中对模型和接口的具体的逻辑实现，这个很好理解

有了这个套路，学习软件设计就有了章法，不会盲目的陷入到软件开发的代码中 [1赞]

作者回复2020-06-02 08:25:20
希望这个结构对你理解软件设计有帮助。

- Jxin 2020-06-01 02:16:23
1.模型：圈定了数据，明确了边界。在我的数据范围内的业务才是我的业务。模型是业务的抽象定义。
2.接口：定义了功能，明确了提供什么服务和这个服务的规格。接口是业务的功能口径。
3.实现：选择技术，明确了功能的性能，满足接口的规格，实现业务的逻辑。实现与业务无关，只考虑接口规格和技术选型。

以上个人理解。课后题有点广。没办法随手解答。得有时间翻个项目才好说。 [1赞]

作者回复2020-06-02 08:15:27
这个课后思考题就是让大家有一个重新思考的过程。

- Kǎfκã²º²º 2020-06-01 00:23:00
模型，通常包含两类要素，一是基本元素，二是这些元素之间的关系。比如常见的CRM，基本元素就包括项目、客户、合同和回款，相互之间的主要关系通常是客户报备，进入立项环节（评估投入产出），再签约，最后进入回款环节。这是基本模型。

这个模型（系统）的接口，就是要为BD提供从客户报备到签约、回款的整个流程管理。

实现就是要考虑如何用消息在这些模块之间传递数据，状态控制、数据查重锁定等等。 [1赞]

作者回复2020-06-02 08:13:21
赞，这个思路很清晰！

- Geek_3b1096 2020-06-02 04:15:50
  喜欢老师只能记住一件事总结风格

  作者回复2020-06-02 08:07:51
  能记住就好。

- 三生 2020-06-01 23:10:07
  分析框架的流程应当从子模型往上建模，
  若从mvc入手，开始分析其内部的子模型是怎么实现，感觉虽然明显看到顶层模型的交互但是会很含糊。
  认为应当自底向上的进行分析，从框架运行的流程（生命周期）进行模型分析再上升为组件模型，最后总结mvc模型/其他模型，这种分析的方式，虽然相反，但是感觉更容易懂